Assurance Cases for Design Analysis of Complex System of Systems Software

Stephen Blanchette, Jr.[*]
Software Engineering Institute, Pittsburgh, PA, 15213

This paper discusses the application of assurance cases as a means of building confidence that the software design of a complex system of systems will actually meet the operational objectives set forth in the project's top-level requirements. The discussion will explain the reasons for selecting the assurance case approach and step through an example of how to apply the method in the described context. The challenges of applying the approach in a real setting will be considered, followed by conclusions. This paper will serve as an anonymous case study. At the close of the paper, readers will have learned how assurance cases have been used in real, non-trivial settings to solve the stated problem. They will have a blueprint for applying similar techniques to their own challenges, with the insight into potential pitfalls that will allow them to succeed.

Nomenclature

k	=	a given key performance parameter

I. Introduction

WHEN one speaks of assurance with respect to software, one can no longer be certain of being understood in the intended context. The phrase *software assurance* has been expanding to include many definitions, ranging from quality, to safety, and, more recently, to security. Arguably, software assurance has been associated with notions of quality (as in *software quality assurance*) longer than with other definitions, but clearly, the quality, safety, and security of software are all related. In that mindset, this paper introduces the idea of applying a technique generally associated with system safety to the problem of assessing software design quality in a complex system of systems. The motivation for doing so is quite plain.

In the aerospace and defense industries, the system of systems paradigm has become all but de rigueur when designing solutions to meet modern problems. While the community lacks a universally accepted definition of the term *system of systems* (often abbreviated as SoS), there do seem to be some distinguishing characteristics. Unlike traditional systems, the components of a SoS typically have function that is independent of the collective. Indeed, that is the power of the SoS paradigm; the individual systems, when working in concert, contribute to capabilities that exceed the simple sum. For example, a single aircraft probably would not qualify as a system of systems; many of its component parts (engines, landing gear, wings, etc.) do little of interest when not assembled together with the airframe. However, a swarm of unmanned aerial vehicles (UAVs), such as was envisioned by the DARPA Joint Unmanned Combat Air Systems (J-UCAS) project, does exhibit SoS behavior. Each individual UAV can be envisioned as having its own sensing and targeting capabilities, but when flown together as a coordinated entity, each individual can extend the reach of those capabilities for the benefit of swarm while also introducing new abilities (such as massed effects, redundant operations, hunter-killer operations, etc.).

The advantages of systems of systems are levels of capability not achievable (at least not practically) in a single, monolithic system. Examples of current SoS development efforts include the Global Earth Observation System of Systems (GEOSS) project, the U.S. missile defense project, the Future Combat Systems (FCS) being developed for the U.S. Army, the U.S. Coast Guard's Deepwater project, and many others. Note a commonality among these SoS examples: the necessity of achieving interoperability among their respective geographically dispersed elements points implicitly to a networking aspect. Networks, especially wireless ones, add a layer of complexity to the SoS domain by limiting the determinism of communications between the SoS components and, in so doing, making predictably of SoS behavior more difficult.

[*] Senior Member of the Technical Staff, Acquisition Support Program, 4500 Fifth Avenue; AIAA Senior Member.

The advent of such complex SoS (sometimes called networks of systems) has given rise to an equally complex problem, namely, assuring that such constructions deliver their required functionality and quality. Often, these end-state qualities are delivered, in part or in full, by software. Yet, traditional testing methods are ill suited to conclusively demonstrating the quality of SoS software. Further, discovering problems at the testing stage has long been known to be one of the least cost-effective times to unearth product quality errors during a software development effort.[1] Late discovery of such defects is even more expensive in a SoS context, both because errors are more difficult to track down in the web of component system interactions and because the ultimate problem resolution may well involve changes to several systems within the SoS.

Ideally, one would like to be able to find SoS software errors no later than design time, when problems are more easily and less expensively resolved. However, traditional design analysis methods, such as architecture evaluations or peer reviews of designs, also struggle in the SoS context. It is easy to miss the subtle interactions among the constituent systems of a SoS because it is difficult to conceptualize its intricacies, while there also is a tendency for engineers to focus on the better defined, and often more easily understood, individual systems. This paper proposes, based on actual experience, that *assurance cases* can be used effectively for reasoning about complex SoS software during design time.

II. What is an Assurance Case?

An assurance case is nothing more than a specialized instance of general case argumentation. To make one's "case," one argues that certain evidence supports a given claim. The basic form is given in Fig. 1. An assurance case, then, simply put, is a structured set of arguments supported by a body of evidence that justifies belief that a given claim is so. As a practical matter, one constructs an assurance case by starting with an overarching claim and then iteratively decomposing it into constituent claims, which, at the lowest level, are supported by direct evidence.

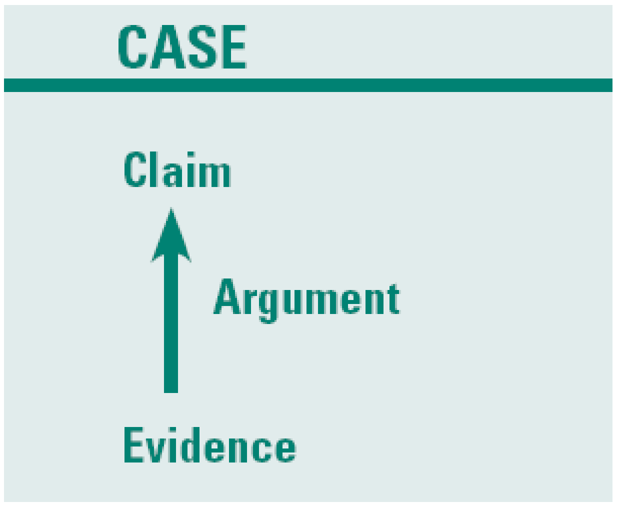

Figure 1. The basic form of case argumentation: Evidence supports the argument that the claim is so.

Applying the assurance case definition to the analysis of complex systems, a c*laim* is just what one would expect: an assertion about some aspect of a system. *Evidence*, typically, consists of documentation (plans, designs, test results, and the like). Note that, ideally, evidence should lean toward the objective, rather than the subjective or, worse, the speculative. For example, a test results document would contain objective evidence of system performance whereas an engineer's opinion about a design's performance characteristics would represent subjective or speculative evidence. In some cases, subjective evidence can support an argument if that evidence also is backed up by objective sources. Subjective evidence also can be used to amplify other objective evidence. Speculative evidence should be avoided. The *argument* in an assurance case links together the sub-claims and evidence in a logical manner, permitting one to draw conclusions about the validity (or invalidity) of the primary claim.[2] Thus, assurance cases allow one to reason about complex systems. Said another way, they allow one to *assure* oneself that certain properties of a complex system hold true.

While assurance cases are a means of determining product quality, they are not a substitute for testing or other methods of product assurance. Rather, they should be applied as part of a comprehensive assurance approach. Indeed, assurance cases can be used as a capstone method that encompasses traditional assurance methods such as testing (in other words, the outputs of those methods become part of the assurance case). Alternatively, assurance cases can be used to augment other assurance methods; they are especially useful in areas where other methods would be too impractical (even impossible) or too expensive. For instance, one can imagine testing the boundary conditions of some safety-critical system (i.e., the points at which such a system is no longer safe). Testing the actual system under such conditions might be hazardous to the testers, while testing a simulation of the system might be very difficult to accomplish with sufficient fidelity.

In fact, many readers will recognize assurance cases as a generalization of techniques used, especially in Europe,[3] to build confidence in the safety of systems. A safety assurance case represents a body of evidence coupled

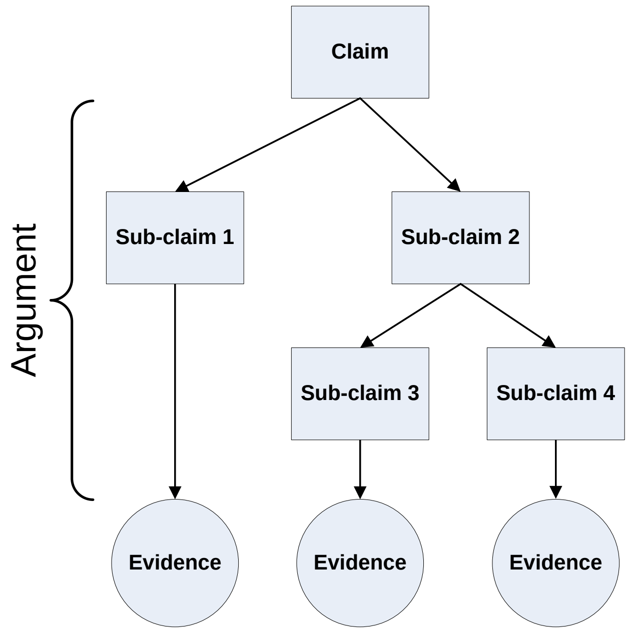

Figure 2. A basic assurance case diagram.

with argumentation that together provide a convincing and valid line of reasoning that a system will be sufficiently safe when used for its intended purpose in its intended environment.[4] Assurance cases, as described here, merely expand on this idea, allowing reasoned arguments to be made about various non-safety aspects of a given system.

Since the goal of using assurance cases in the current context is to help understand the design characteristics of complex SoS software, having a notation for these structured arguments is quite useful. Fortunately, assurance cases can be diagramed rather easily.[†] Figure 2 demonstrates the concept. At the top of the figure is the main claim, the thing about which we would like to reason. This claim is decomposed into (or *solved by*) two sub-claims (numbered 1 and 2). Sub-claim 2 is further broken down into two additional sub-claims, 3 and 4. Each of the leaf level sub-claims (numbers 1, 3, and 4) is supported by evidence. One then argues up: The evidence supports the sub-claims, which in turn support the higher-level sub-claims, which in turn support the main claim. Thus, one has an argument, substantiated by evidence, that allows one to affirm (or refute) the main claim.

III. Application to the Analysis of a Complex SoS Software Design

The broader application of assurance cases has been studied in a number US Department of Defense (DoD) projects.[5] One of these, a project hereafter referred to as *Project X* for the sake of anonymity, needed a way to understand whether the software being built would be sufficient to implement the SoS functionality necessary for project success, and it needed to know before a large portion of that software would be developed; due to long lead times on manufacturing, decisions to proceed with the production of constituent systems would have to be made early. The question became, how could project management, stakeholders, and decision makers know that the SoS software design would meet operational needs?

The simplicity of that question belies its complexity. As is the case with many SoS projects, *the* software design was actually *many* designs, documented in many places. In addition, the designs were not yet complete; indeed, many of the final designs would not be complete until well after production decisions had to be made. Further, there was the question of how to characterize the operational needs in terms that would be meaningful at the software level. To these complications was added the desire to base conclusions as much as possible on actual data rather than on optimistic plans and confident assertions.

The size of the analysis space and the desire to use a data-driven method suggested an assurance case approach, although two constraints weighed heavily on the ability to complete the task:

1) Time. Using assurance cases was a rather late thought; results needed to be available as input into a project-level review that had a hard deadline.
2) Resources. The team available to do the work was small (only four people initially, none of them working full time on Project X). Further, within the team, domain experience varied widely.

These constraints prevented the sort of methodical, in-depth investigation typical of safety case analysis. Instead, the analysis team had to focus on finding the big, "show-stopper" type of problems rather than trying to identify every potential design issue. With the effort thusly bounded by programmatic realities, it was possible to address some of the other analysis challenges.

[†] A subtle distinction is that the actual "case" is the logic of the argument itself. The diagram is a (hopefully faithful) depiction of that logic. See Ref. 6.

Because the overall SoS software designs were not complete, the goal of using data as the foundation of the analysis was somewhat challenging. Since the software was being implemented in a series of builds, evidence would have to consist of objective results from tests and experiments executed against completed builds, combined with the analysis team's subjective evaluations of design artifacts for the current build and plans for the remaining builds.

Gathering the required data was often a challenge due to the vast quantities of information available and its wide distribution throughout the project. The need to access in-progress work further exacerbated the problem, since such material often was not generally available to project personnel. The best solution to these challenges was strong management advocacy. Project management facilitated finding the right people with whom to discuss data needs and provided the necessary approvals to gain access to information that was not yet published.

The final challenge to be addressed was how to characterize operational needs in terms of software. The top-level SoS requirements were expressed in terms relevant to the military users, not to software developers. A detailed study of the requirements would be time consuming and would largely repeat the work already done by systems engineers, software developers, and testers. In addition, simply relying on requirements traceability from the highest level down to the software was not a good solution; errors in traceability had the potential to skew the entire analysis, and issues with the requirements database were a well-known, ongoing project challenge.

In the end, the analysis team decided against trying to re-cast the operational needs of Project X in terms of software. Instead, the team analyzed the software contributions to the definitive characterization of operational needs – the SoS *key performance parameters* (KPPs). Within the DoD, KPPs are the system characteristics essential for delivery of an effective military capability.[7] All DoD projects have some number of KPPs to satisfy in order to be considered acceptable from an operational perspective. For example, any DoD system that must send or receive information externally is required to fulfill the Net-Ready KPP (NR-KPP).[8]

Given this new perspective, the way to apply assurance cases became clear: demonstrate that the software design supported each KPP. Thus, the chief claim for any given KPP then took the form of

$$\text{The SoS will satisfy KPP } k \tag{1}$$

where k represents the specific KPP being examined. To make the analysis clear, each main claim was more detailed, usually repeating the actual text of the KPP rather than simply identifying it by name. The form of the claims, however, was analogous to that shown here.

Having decided upon the KPP approach, the initial stages of the analysis resembled a structured decomposition of each relatively abstract KPP into more precise statements that could be more readily assessed in terms of evidence. The challenge was to break down the KPPs into sub-claims in a logical and consistent manner but without accumulating so much detail as to overwhelm the analysis. At some point, analysts had to use their best engineering judgment to make a bit of a leap from higher-level systems concepts to lower-level concepts that were directly supported by software. Since the intent of the analysis was to understand the SoS software, rather than each KPP in detail, the leap was justified.

IV. An Example

A good way to understand the application of assurance cases in the manner described above is to step through an example. In this section, an example of applying assurance case analysis to a KPP from a SoS software perspective is examined. The example is an analysis of a software design's contribution to satisfying the Net-Ready KPP, as defined in Ref. 8, for a hypothetical SoS project. The analysis has been simplified in the interests of demonstrating the technique rather than illustrating a comprehensive result.

The example is bounded as follows. The hypothetical SoS will be distributed, so that its constituent pieces must communicate over a network. (Obviously, it must be able to communicate with other systems, or the NR KPP would not apply.) It is being developed in multiple builds, and the core communications architecture needed to support the network has already been implemented in the initial build. Software plays a significant role in the communications and networking functions, including security and information transfer (not at all unreasonable assumptions). Security (including information assurance) is being implemented over multiple builds (this is often the case on real projects due to the difficult and lengthy certification processes and a fair degree of uncertainty about the full scope of requirements in these areas). Lastly, while some quality of service (QoS) measures can be estimated through models and simulations, there still will be some level of uncertainty until all message traffic has been implemented, the fully realized network (including software and hardware) with all expected nodes is in place, and actual QoS can be demonstrated.

A. Notation

Figure 3 shows the notation used in the remaining diagrams for the example. The assurance case diagrams described in this paper generally follow goal structuring notation (GSN), which is a technique for graphical

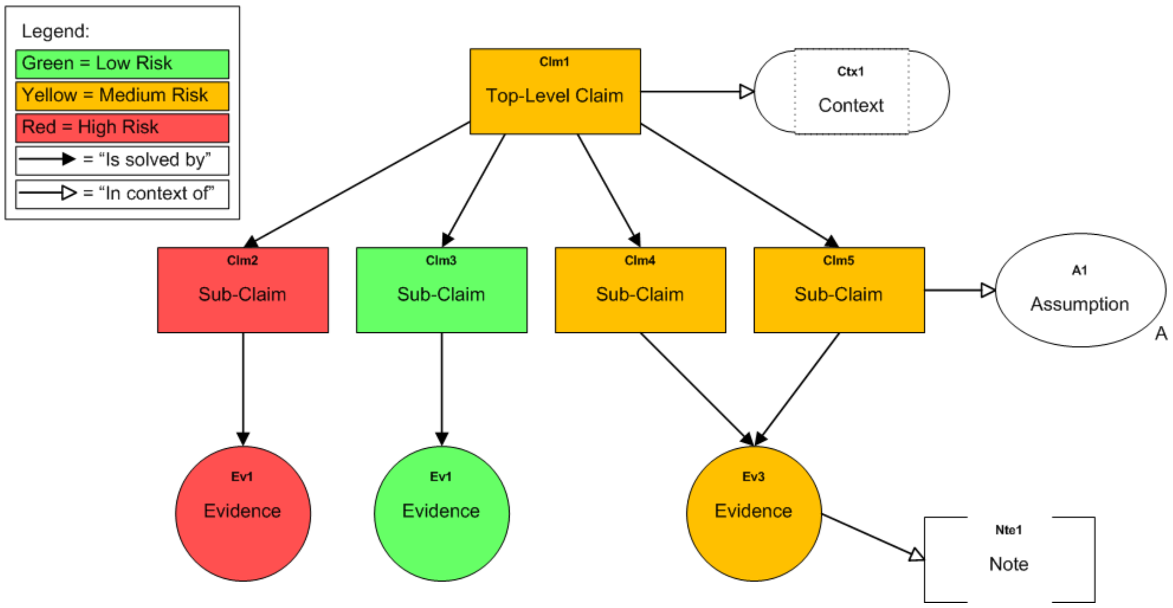

Figure 3. Assurance case notational elements shown with legend.

reasoning.[‡] The rectangular boxes represent claims. The topmost claim is the main claim being reasoned about, while the sub-claims comprise the overall argument. The circles represent evidence and appear at the lowest level of each branch in the argument, directly supporting the lowest level claims. The remaining symbols provide additional exposition with respect to the claims or evidence. Oblongs (the bathtub shapes) represent context. To aid a reader's understanding, context is useful to further elucidate a point or to reference external sources of information. Ovals subscripted with the letter A represent assumptions. It is often valuable to document explicitly any important assumptions being made in an argument, since an incorrect assumption can invalidate an argument. The broken boxes represent notes, which can be used to provide relevant information not directly related to the argument. The final elements of note are the arrows, which establish the relationships among the other elements. An arrow with a filled head denotes an *is solved by* relationship, as in "Claim 1 is solved by sub-claim 2." The filled arrow is always used to show relationships between claims or between claims and evidence. An arrow with an empty head shows an *in context of* relationship; these arrows are used to connect claims or evidence to the ancillary symbols used for context, assumptions, and notes.

One can use most any drawing program to produce assurance case diagrams conforming to GSN, although there are purpose-built assurance case tools available that provide additional niceties (e.g., better support for very large diagrams, consistency checking mechanisms, databases to capture details regarding each node, etc.) over generic drawing packages. For instance, diagrams in this example were created with the Assurance and Safety Case Environment (ASCE) tool from Adelard, LLP.

B. Analysis

The base claim for the example is *the system of systems supports Net-Centric military operations*. Figure 4 [next page] depicts the overall assurance case diagram[§].

[‡] Interested readers can consult Ref. 9 for additional information regarding GSN.

[§] The diagram is a tree structure, and the nodes in it are numbered top-down and left-to-right, consistent with a depth-first, preorder traversal, but with each node type having its own numbering system (e.g., claim nodes are numbered 1-*n*, evidence nodes are numbered 1-*m*, etc.). Although other structures are possible (an indented list, for instance), the tree structure reinforces the notion of a decomposed argument.

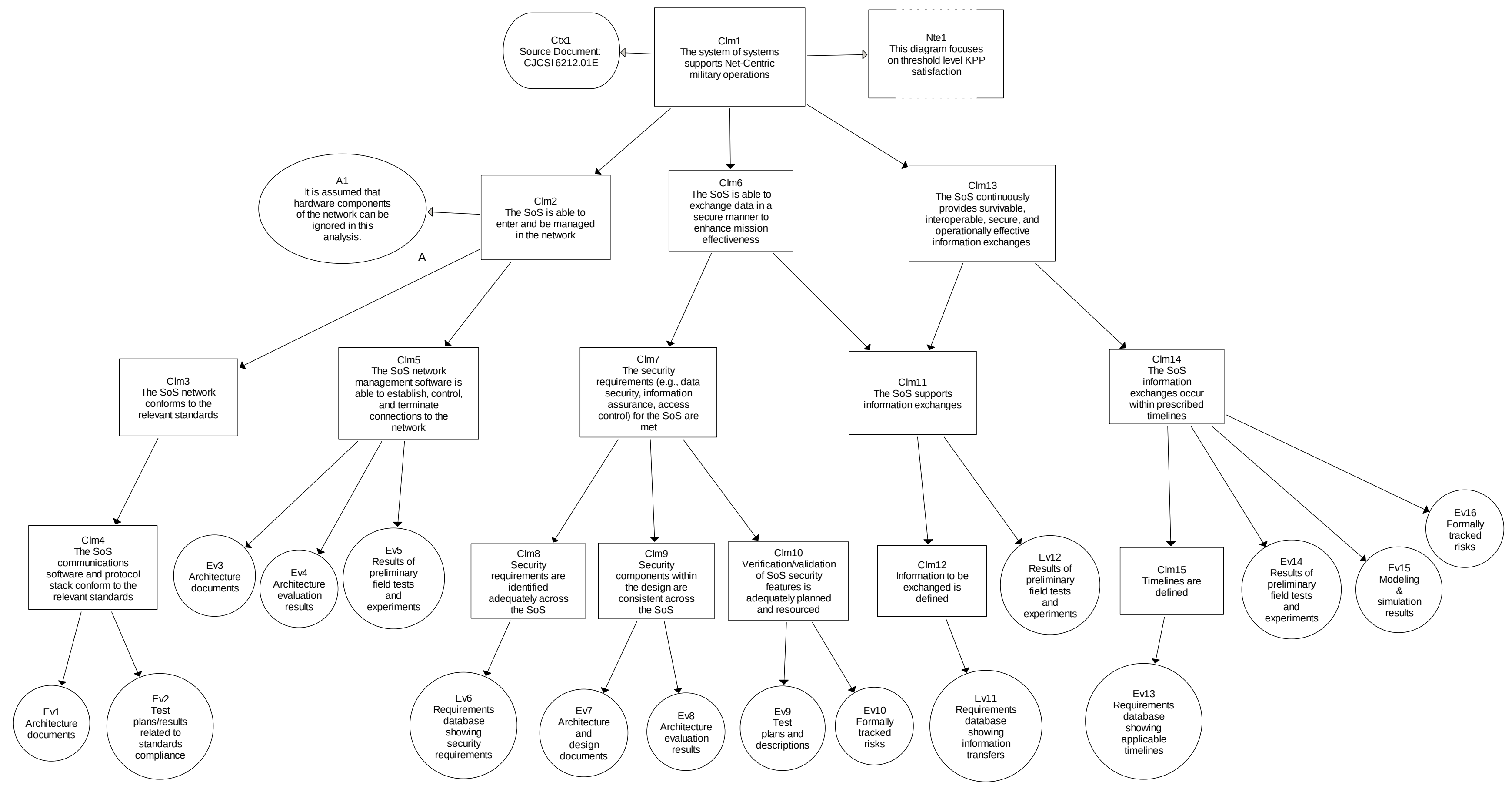
Ctx1 Source Document: CJCSI 6212.01E
Clm1 The system of systems supports Net-Centric military operations
Nte1 This diagram focuses on threshold level KPP satisfaction
A1 It is assumed that hardware components of the network can be ignored in this analysis.
A
Clm2 The SoS is able to enter and be managed in the network
Clm6 The SoS is able to exchange data in a secure manner to enhance mission effectiveness
Clm13 The SoS continuously provides survivable, interoperable, secure, and operationally effective information exchanges
Clm3 The SoS network conforms to the relevant standards
Clm5 The SoS network management software is able to establish, control, and terminate connections to the network
Clm7 The security requirements (e.g., data security, information assurance, access control) for the SoS are met
Clm11 The SoS supports information exchanges
Clm14 The SoS information exchanges occur within prescribed timelines
Clm4 The SoS communications software and protocol stack conform to the relevant standards
Ev3 Architecture documents
Ev4 Architecture evaluation results
Ev5 Results of preliminary field tests and experiments
Clm8 Security requirements are identified adequately across the SoS
Clm9 Security components within the design are consistent across the SoS
Clm10 Verification/validation of SoS security features is adequately planned and resourced
Clm12 Information to be exchanged is defined
Ev12 Results of preliminary field tests and experiments
Clm15 Timelines are defined
Ev14 Results of preliminary field tests and experiments
Ev15 Modeling & simulation results
Ev16 Formally tracked risks
Ev1 Architecture documents
Ev2 Test plans/results related to standards compliance
Ev6 Requirements database showing security requirements
Ev7 Architecture and design documents
Ev8 Architecture evaluation results
Ev9 Test plans and descriptions
Ev10 Formally tracked risks
Ev11 Requirements database showing information transfers
Ev13 Requirements database showing applicable timelines

Figure 4. Assurance case diagram for the hypothetical SoS.

As shown in Fig. 4, the main claim is decomposed into three sub-claims:

- *The SoS is able to enter and be managed in the network* (Claim 2)
- *The SoS is able to exchange data in a secure manner to enhance mission effectiveness* (Claim 6)
- *The SoS continuously provides survivable, interoperable, secure, and operationally effective information exchanges* (Claim 13)

This first division is straightforward; the claims correspond with the definition of the NR KPP, which states that a system

> ...must support Net-Centric military operations. The…system…must be able to enter and be managed in the network, and exchange data in a secure manner to enhance mission effectiveness. The…system…must continuously provide survivable, interoperable, secure, and operationally effective information exchanges to enable a Net-Centric military capability.[8]

Each of the three principal sub-claims and their sub-trees is discussed in turn.

1. Network Entry and Management

Claim 2 is divided into two sub-claims numbered 3 and 5, respectively.

Claim 3 states that the SoS network conforms to relevant standards. The logic behind the claim is that in order to be able to enter and be managed in the network, the SoS must first implement a set of standardized networking protocols, just as a Web application would have to implement the Internet Protocol (IP) in order to be useable. This claim is still at too high a level to be tied to the relevant software, so an additional sub-claim, Claim 4, is needed. Claim 4 states that the SoS communications software and protocol stack conform to the relevant standards. From here, evidence can be gathered to support Claim 4. The evidence identified in Fig. 4 includes the relevant software architecture documents, test plans, and test results to demonstrate that the applicable standards had been identified in the design and that their implementation was checked for and successfully demonstrated. Since the analysis was done for a hypothetical SoS, the details of the evidence are sketchy; for a real system, one would identify the specific name, title, and release date of any documentation cited as evidence to support or refute a claim.

Claim 5 proceeds along a similar pattern. The claim states that the SoS network management software is able to establish, control, and terminate connections to the network—all obviously necessary functions for being able to operate within a network environment. Here, the claim is clearly related to software and no further decomposition is necessary. The evidence selected to support this claim includes architecture documents, the results of architecture evaluations, and the results of preliminary field tests and experiments. The architecture should describe the essential functions related to network connections. The architecture evaluation results should demonstrate that the designed network management features would support the necessary usability aspects of network management. The field test results should demonstrate the adequacy of the implementation. Note that a conscious decision has been made to ignore traditional verification and validation results (such as results from integration testing) for the network management software in favor of the (presumably more extensive and robust) field test results. For networking software, testing under conditions closest to the actual environment and loads will likely yield the most realistic assessments of the software's performance. If actual conditions can be faithfully produced in a laboratory environment, then using integration test results in the analysis would be reasonable.

2. Secure Data Exchanges

Claim 6 is decomposed into two sub-claims numbered 7 and 11, respectively.

Claim 7 states that the security requirements (e.g., data security, information assurance, access control, etc.) for the SoS are met. Clearly, this must be true if Claim 6 is to be true. Much is implied by Claim 7, necessitating a further decomposition into Claims 8, 9, and 10. Claim 8 states that the security requirements are identified adequately across the SoS. This claim can be directly supported by evidence, such as the requirements database (or documents, if a database is not used). To be complete and sufficient, the requirements would have to be decomposed to all relevant components of the SoS and traceability (forward and backward) would have to demonstrate that all requirements were successfully flowed down without any extras (i.e., lower level requirements that are unjustified by the top-level requirements) added. Claim 9 states that the security components within the design are consistent across the SoS. Here again, the best evidence to support the claim would be the relevant architecture documents, coupled with the results of the corresponding architecture evaluations. Claim 10 states that verification and validation of the SoS security features is adequately planned and resourced. For this claim, test plans and descriptions have been chosen as the evidence, since the initial assumptions were that security requirements had not yet been implemented. Also included as evidence are some formally tracked risks. Experience with many SoS projects suggests that complying with the applicable directives related to security, as well as achieving the necessary external certifications and gaining authority to operate on the network, is a long and arduous path strewn with potholes. Thus, one expects the staff of the hypothetical SoS to have at least one risk related to security that they are attempting to mitigate.

Claim 11, the SoS supports information exchanges, also must be true if those exchanges are to be handled securely as stated in Claim 6. For this claim, a combination of further decomposition and direct evidence has been used. First, the information to be exchanged must be defined; Claim 12 says that, and can be supported by the requirements database as evidence much the same way that Claim 8 was supported (as discussed in the previous paragraph). The direct evidence to support Claim 11 is the results of preliminary field tests. Presumably, at least a subset of the required data exchanges would have been exercised as part of the network checkout in the early build of the hypothetical SoS.

Note that Claim 11 also supports the third principal sub-claim, Claim 13.

3. *Continuous Data Exchanges*

Claim 13 states that the SoS continuously provides survivable, interoperable, secure, and operationally effective information exchanges. In order to do that, the SoS must first support information exchanges. Claim 11, discussed in the previous section, covered that notion. For brevity of discussion, it is assumed that Claim 6 covers the survivable and secure aspects of the claim, and that Claim 2 covers the interoperable aspect (after all, it would not be much of network unless the hypothetical SoS could both send and receive information). The last remaining aspect, “operationally effective,” implies time. Specifically, as captured by Claim 14, the SoS information exchanges occur within prescribed timelines. Obviously, one first wants to ensure that the timelines are defined. Claim 15 says that, and the requirements database will once again serve as the source of evidence. In addition to timeline definitions, though, there should be direct evidence of those timelines being met to support Claim 14. For such evidence, the results of preliminary field tests are once again called upon, since field tests would supply the most realistic conditions for demonstrating timeliness of information exchange. It is also assumed that some degree of modeling and simulation (M&S) has been done to predict network performance, since it will difficult to adequately test all aspects of network quality of service (QoS) until all elements of the SoS have been completed. Formally tracked risks are the last piece of evidence to support Claim 14. Experience suggests that there will be a delta between M&S results and the results of preliminary field tests. Such a performance delta would represent uncertainty about meeting the required timelines and should be captured as one or more risks, which project personnel should be attempting to mitigate.

C. A Word about Evidence

When the time comes to seek out and examine evidence, independent analysts should consult with project personnel who have expertise in the technical area(s) of interest. Often, and especially so for SoS projects, evidence to support a given claim is distributed throughout several documents, and finding all the relevant ones requires an insider’s knowledge of the project. Another common occurrence is an analyst will expect to find a specific type of document, say, a schedule, only to discover that no one on the project knows of the existence of such a document; instead, project staff use a Gantt chart or IMS (for integrated master schedule, pronounced as a word rather than spelled out). To avoid misunderstandings due to differences in technical dialects, it is important for analysts to explain the type of information they are looking for rather than to simply ask for a specific document.

D. Scoring

When performing an assurance case analysis of a completed design, the outcome is rather black-and-white: either design artifacts are complete and sufficient, or they are not. Reviewing an in-progress design requires a more nuanced approach, one that reflects relative risk, since the design artifacts will necessarily be in different stages of completion. For this example, a simple and familiar “stoplight” approach (so named for the red-yellow-green coloring) to scoring is taken, where the color red designates a relatively high risk area, the color yellow designates a relatively medium risk area, and the color green indicates a relatively low risk area. The rules for assigning colors are slightly different at the evidence level than they are at the level of the claims, as is shown in Table 1.

Table 1. **Scoring Rules for the Example**

For Evidence	
Green	Evidence is complete and adequate
Yellow	Evidence is incomplete or planned for the future
Red	Evidence is complete but inadequate, planned but now late, or non-existent
For Claims	
Green	All lower-level claims and supporting evidence are green
Yellow	Some lower-level claims and supporting evidence are a combination of yellow and red
Red	All, or an overwhelming majority of, lower-level claims and supporting evidence are red

Note that the application of the Red and Yellow rules to claims is subjective in situations where sub-elements are not all uniformly red or yellow: An analyst must decide what constitutes an "overwhelming majority" in order to determine if a higher-level claim should inherit a red color or yellow color (and corresponding level of risk). Any number of alternate scoring schemes is possible, and it is for each project to determine the permissible degree of quantification or subjectivity.

By making the following assertions about the evidence that was identified in the hypothetical SoS example:

1) only a subset of information exchanges has been implemented to date;
2) the noted risks are, at best, medium at this time;
3) the security architecture has not been completely propagated across the SoS;
4) additionally, an evaluation of the security architecture revealed some design choices that will prevent system accreditation; and
5) preliminary field tests indicate some information exchanges are exceeding prescribed timelines for completion,

and then applying the rules in Table 1, the overall analysis of the example would be medium risk; that is, there would be a medium risk of the hypothetical system of systems *not* supporting Net-Centric military operations. The overall analysis tree would appear as shown in Fig. 5 [next page], and would provide project managers with a sort of roadmap for prioritizing and addressing the issues.

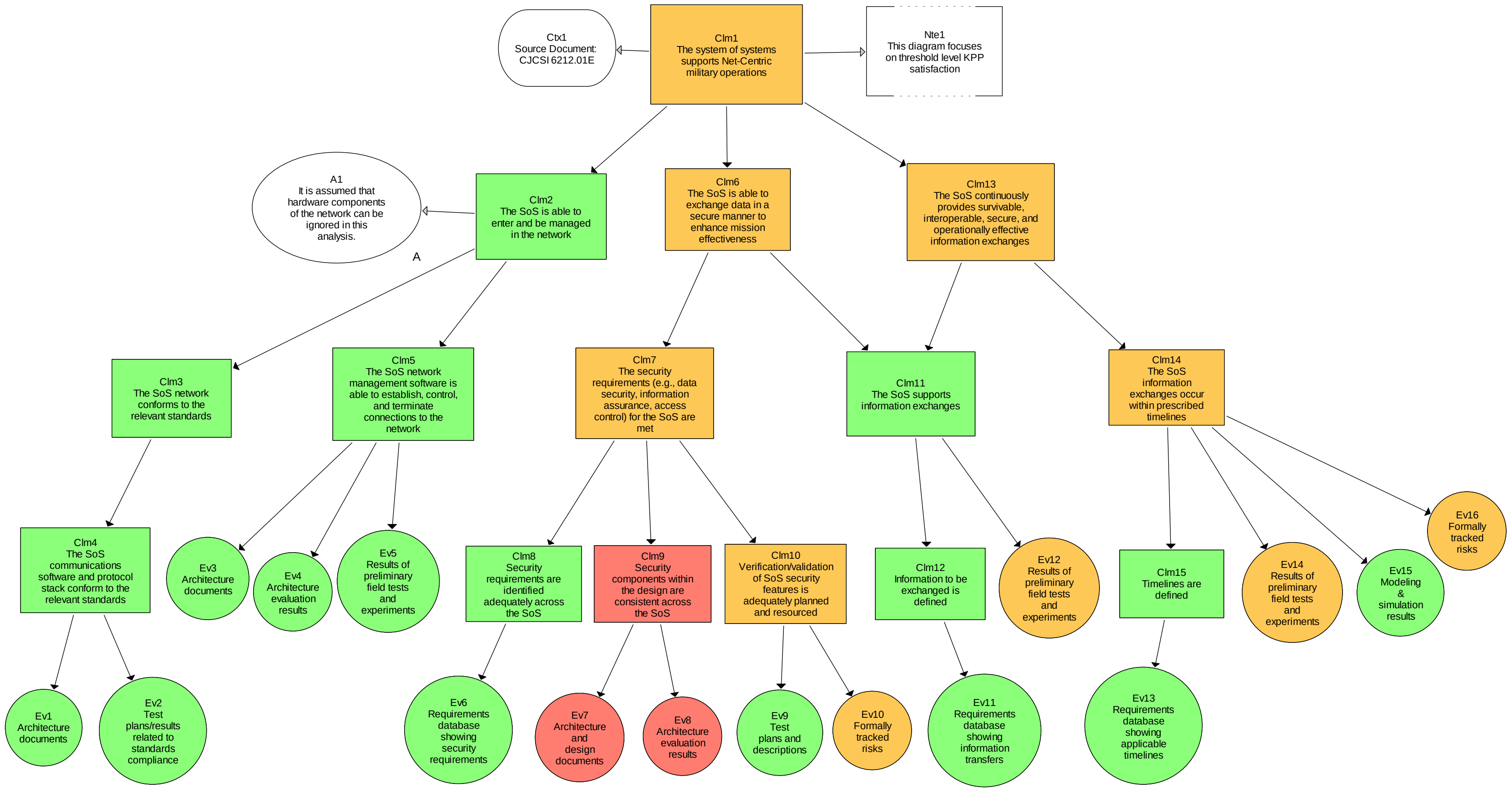
Ctx1 Source Document: CJCSI 6212.01E
Clm1 The system of systems supports Net-Centric military operations
Nte1 This diagram focuses on threshold level KPP satisfaction
A1 It is assumed that hardware components of the network can be ignored in this analysis.
A
Clm2 The SoS is able to enter and be managed in the network
Clm6 The SoS is able to exchange data in a secure manner to enhance mission effectiveness
Clm13 The SoS continuously provides survivable, interoperable, secure, and operationally effective information exchanges
Clm3 The SoS network conforms to the relevant standards
Clm5 The SoS network management software is able to establish, control, and terminate connections to the network
Clm7 The security requirements (e.g., data security, information assurance, access control) for the SoS are met
Clm11 The SoS supports information exchanges
Clm14 The SoS information exchanges occur within prescribed timelines
Clm4 The SoS communications software and protocol stack conform to the relevant standards
Ev3 Architecture documents
Ev4 Architecture evaluation results
Ev5 Results of preliminary field tests and experiments
Clm8 Security requirements are identified adequately across the SoS
Clm9 Security components within the design are consistent across the SoS
Clm10 Verification/validation of SoS security features is adequately planned and resourced
Clm12 Information to be exchanged is defined
Ev12 Results of preliminary field tests and experiments
Clm15 Timelines are defined
Ev14 Results of preliminary field tests and experiments
Ev15 Modeling & simulation results
Ev16 Formally tracked risks
Ev1 Architecture documents
Ev2 Test plans/results related to standards compliance
Ev6 Requirements database showing security requirements
Ev7 Architecture and design documents
Ev8 Architecture evaluation results
Ev9 Test plans and descriptions
Ev10 Formally tracked risks
Ev11 Requirements database showing information transfers
Ev13 Requirements database showing applicable timelines

Figure 5. Assurance case diagram for the hypothetical SoS showing colored scores.

V. Lessons Learned in Applying Assurance Cases

It may seem, having stepped through the simplified example, that applying assurance cases to a real project is a straightforward matter. However, potential adopters of this approach should be mindful of several lessons learned.

On Project X, the constraints on time and resources to perform the analyses forced the team to focus on big picture risks rather than performing the more in-depth analysis supported by the method. While this approach was still valuable given Project X's stage in the development process, it is yet to be seen if a more thorough analysis could have yielded additional critical risks or identified additional strong points in the design. In any case, these types of analyses should be part of project planning from the outset, so that conscious decisions can be made about the level of resources to apply based on the desired outcomes.

Another good reason for up front planning is to help communicate the effort among affected areas of the project. Conceiving of the assurance case analysis mid-stream in the project and planning it from the ground up necessitated repeated briefings to management as well as explanations each time a different project group had to be queried for information. In addition to the obvious inefficiency of proceeding in that manner, some project groups still will be reluctant to share data absent direction to do so by their management. Making the analysis a planned part of the program from the outset would ameliorate at least some of these difficulties.

The project strongly valued having an independent team perform the analyses. Nevertheless, there were tradeoffs made implicitly in the decision to use an independent team. For one, the time constraints necessitated using the independent experts already available to the project, rather than bringing in a new and highly qualified analysis team. While all team members were skilled in software engineering, few had any experience with assurance cases, which created a learn-by-doing environment. Additionally, although members of the analysis team had been advising Project X on software issues, most suffered at least some lack of knowledge about the operational domain of the SoS, making it difficult at times to bridge the gap from operational-level requirements to the software areas that supported them. These factors slowed progress and contributed to the schedule compression of the task. Here, too, advance planning would have helped; it could have been used to allow for training in the analysis method and in the basic concepts of the operational domain or even to arrange for different (or additional) experts to be available. Another solution might be to have project personnel perform the analysis, which would eliminate the domain knowledge issues (but also introduce potential subjectivity into the analysis).

Starting the analyses at a level as high as the KPPs caused diagrams to grow quite busy very quickly. There is an enormous tendency to make very small steps in logic as one progresses down the analysis path in order to ensure the absolute soundness of the argument. Indeed, this approach should be preferred in most cases to ensure the soundness of the results. However, when dealing with a vast analysis space and limited time, one must balance between precision in argument and comprehensibility. At some point, adding more detail is of diminishing value and judicious choices must be made about what to represent and what to omit.

Also contributing to diagram busyness is ancillary information about claim nodes. While the notation allows for adding various notes about context and assumptions, in practice these elements serve to clutter the diagram in all but the simplest cases. If the diagram is viewed using the tool with which it was created, the clutter effect is not too serious; a user can enlarge or reduce areas of interest. For general presentation and reporting purposes, however, diagram clutter presents a serious difficulty in conveying an understandable message. An ameliorating approach that seems to work well (when using the ASCE tool) is to capture assumptions and other incidental information within the node being described rather than separately. This approach is a minimal inconvenience to the analysts, who must then open a node to see the information. However, it is a huge benefit to those for whom the analysis is being performed; the logic of the analysis is clear and clean, without extraneous material. For managers who wish to see the detailed information, it can be captured readily enough in report appendices or even as speaker notes during out-briefs.

For any given SoS project, not all of the KPPs will necessarily have a software component. Performing an initial evaluation of each KPP to determine if software will play a role is an important step to avoid spending time on a detailed analysis that will ultimately come to a dead end.

VI. Observations

For a very large and complex SoS, the assurance case method worked well as a way of analyzing the software design. It helped bring order to a fairly nebulous analysis task and, perhaps more importantly, it provided a framework that allowed the analysis team to select among innumerable artifacts for potential study. It should be clear that the approach is not reliant upon KPPs; one could easily start at lower levels of requirements to begin the software analysis. It should also be clear that the approach could scale down to focus on a specific technical point (although there may be some lower limit at which the investment of time and resources is not justified by the gain received).

Four to six weeks to complete an analysis thread is likely a reasonable estimate for a competent evaluator with sufficient domain knowledge and access to the artifacts, with most of that time devoted to finding, accessing, and reviewing the artifacts used as evidence. The method itself is not difficult to learn. Applying the method is not especially difficult, although the length of time to develop an argument structure will vary with an analyst's familiarity with the domain. As one would expect, finding the relevant pieces of evidence takes relatively less time for project personnel than for independent experts. However, even with support from project personnel, independent analysts will likely take somewhat longer to find and gain access to the necessary artifacts. Of course, the time needed to review artifacts will vary with the depth of the analysis, the volume of artifacts to review, and the level of domain experience of the reviewer.

VII. Conclusion

This paper has demonstrated the use of assurance cases for analyzing an in-progress software design on a large, complex system of systems project. The technique has been used to analyze software designs of actual SoS projects with some success. Given the difficulty of testing or even understanding a SoS software design in its entirety, the usefulness of assurance cases becomes apparent: assurance cases allow engineers and managers to reason about the SoS under construction and to find potential problems early in the development cycle when they are generally less costly to remedy.

Ideally, one would like to plan the use of assurance cases into the project from the beginning in order to ensure the necessary resources will be available at the appropriate time and sufficient allowances have been factored into the schedule. With some effort, however, it is possible to insert the technique as an unplanned task. If the analysis is not preplanned, compromises in the level of detail may be necessary in order to stay within schedule and budget constraints.

The method can be trained easily enough to experienced engineers. Experience is important; evaluators must be savvy enough to ask the right questions about the design lest there be critical gaps in the analysis. To be most effective, the team performing the analysis should be familiar with the technical domain of the project. Whether the team is composed of project personnel, independent consultants, or a combination is a project decision.

Clearly, the method described here can be applied with varying degrees of rigor but the more rigor desired, the more time and resources must be allowed for the execution of the technique. Increasing rigor also implies increasing complexity of the analysis, so some thought should be given to how, and to whom, the results are to be presented to ensure the message is not lost in the details.

Experience with actual projects suggests that the assurance case technique is a powerful tool for analyzing a large and complex SoS software design. It provides a means of taking a crosscutting look at a SoS, a perspective often achieved only with great effort even in less complex development projects. Assurance cases give managers answers about design progress that are demonstrably rooted in facts and data instead of opinions based on hope and best intensions.

Acknowledgments

S. Blanchette thanks Charles Weinstock, John Goodenough, and Marilyn Phillips of the Software Engineering Institute and Linda Esker of the Fraunhofer Center at the University of Maryland for their contributions to the work that led to this paper.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University[**] for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

[**] NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

References

[1]Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Englewood, NJ, 1981, pp. 39-41.

[2]Emmet, L., and Guerra, S. “Application of a Commercial Assurance Case Tool to Support Software Certification Services,” *Proceedings of the 2005 Automated Software Engineering Workshop on Software Certificate Management (SoftCeMent'05)*, Association for Computing Machinery, New York, 2005, pp. 51-55.

[3]Habli, I.; Kelly, T., "Safety Case Depictions vs. Safety Cases - Would the Real Safety Case Please Stand Up?" *Proceedings of the 2nd Institution of Engineering and Technology International Conference on System Safety*, Institution of Engineering and Technology, London, 2007, pp.245-248.

[4]Kelly, T., and Weaver, R., “The Goal Structuring Notation: A Safety Argument Notation,” IEEE Computer Society, 2004.

[5]Bloomfield, R. E., Guerra, S., Miller, A., Masera, M., and Weinstock, C. B., “International Working Group on Assurance Cases (for Security),” *IEEE Security & Privacy*, Vol. 4, No. 3, May/June 2006, pp. 66-68.

[6]Ellison, R., Goodenough, J., Weinstock, C., Woody, C., “Survivability Assurance for System of Systems,” Software Engineering Institute, Rept. CMU/SEI-2008-TR-008, Pittsburgh, PA, May 2008.

[7]Chairman of the Joint Chiefs of Staff Instruction, “Joint Capabilities Integration and Development System,” CJCSI 3170.01F, 2007.

[8]Chairman of the Joint Chiefs of Staff Instruction, “Interoperability and Supportability of Information Technology and National Security Systems,” CJCSI 6212.01E, 2008.

[9]Kelly, T. P., “Arguing Safety – A Systematic Approach to Managing Safety Cases,” Ph.D. Dissertation, Computer Science Dept., Univ. of York, York, England, UK, 1998.